University of Bahrain
College of Information Technology
Department of Computer Science
First Semester, 2010-2011
ITCS215 (Data Structures)

Final Exam

Question 1 [12 Marks]
Write a member function **validate** to be included in class **linkedListType** that will check that the first half of the linked list is not equal to the second half. If they are equal, then delete either the first half or the second half of the list. Assume that the number of elements in the list is even or the list is empty.
Note: You are not allowed to use any member function of class **linkedListType**.
Function prototype:

```
void validate( );
```

Question 2 [12 Marks]
Write a member function called **pushWithNoDuplicates** to be included in class **stackType** that accepts an **item** of type **Type** as parameter and inserts the **item** at top of the stack if the **item** is not already in the stack.
Function prototype:

```
void pushWithNoDuplicates (Type& item);
```

Note that the class **stackType** has data members **list**, **maxStackSize** and **stackTop**.

Question 3 [12 Marks]
Write a non-member function called insertAt that inserts an item at a given position in an array-based queue q1. The function prototype is:

```
bool insertAt(const queueType<Type>& q1, const Type&  item, int
position);
```

Note that the **position** of the front element of the queue is 0 and increases by 1 for each subsequent element. The function returns false, if the position is invalid i.e., insertion attempt is before the queueFront or after the queueRear, otherwise it returns true.

Use common queue operations only, such as:
isEmptyQueue(), isFullQueue(), addQueue(const Type &), deleteQueue(), front(), the overloaded assignment operator = and the copy constructor function.

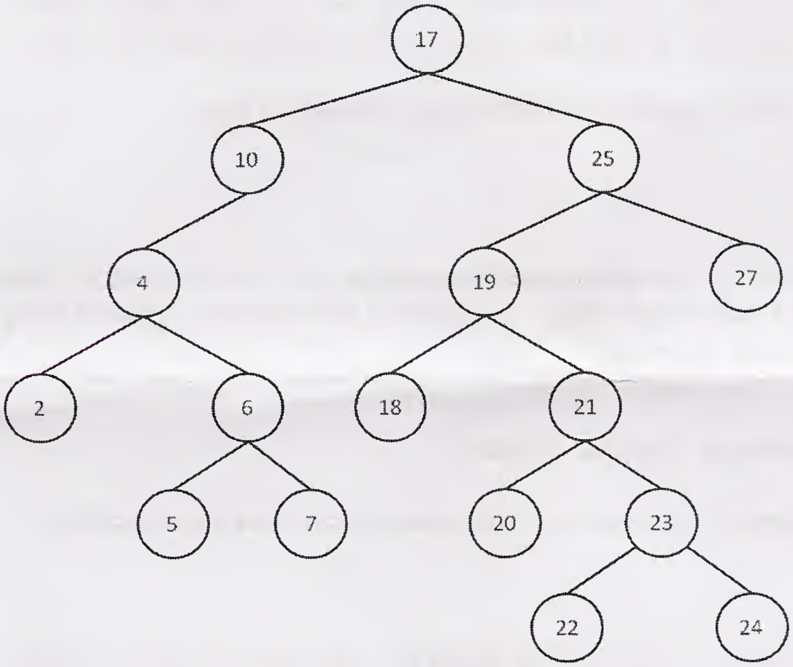
You may use local objects of type `queueType` in your function. Assume that q1 contains less than 20 items.

Examples:

q1(before): A B C C H A I	q1(before): A B C C D H J I	Q1(before): A B D C H J A I
position: 2, item: **X**	position: -1, item: **X**	position: 11, item: **X**
q1(after): A B **X** C C H A I	q1(after): A B C C D H J I	Q1(after): A B D C H J A I
return true	return false	return false

Question 4 [10 + 8 Marks]

(A) Consider the following binary search tree:



(i) **[5 Marks]** Redraw the binary search tree after deleting the nodes with `info` 10 and 25.

(ii) **[5 Marks]** Redraw the binary search tree after inserting the keys 3, 26, 15 consecutively in the original binary search tree.

(B) [8 Marks] Write a recursive private member function called **`sumKeys`** to be included in class `binaryTreeType`. The function returns the sum of the `info` of all the nodes in a binary tree. Assume that the nodes of the binary tree contain numbers as the `info`.

This function is called from a public member function `treeSumKeys`, given as follows:

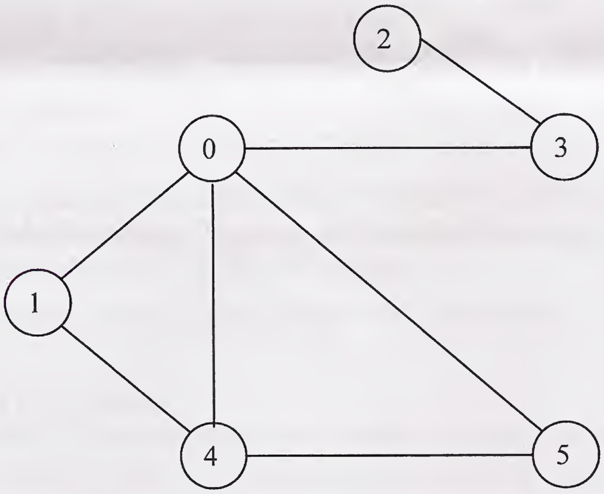
```
template<class Type>
Type binaryTreeType<Type>::treeSumKeys()
{
     return sumKeys(root);
}
```

Function Prototype:

```
   Type sumKeys(nodeType<Type> *p);
```

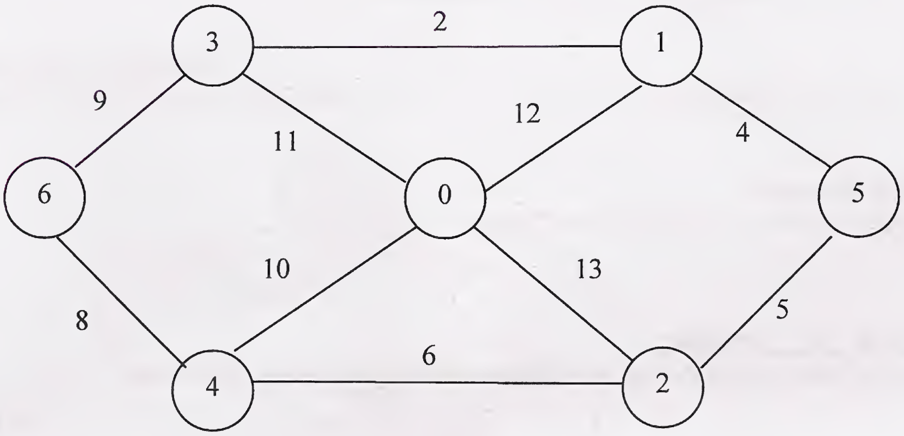
Question 5 [5 + 5 + 8 Marks]

(A) For the following graph, find the **adjacency list** representation of the graph:



(B) For the graph shown in (A), find the sequence of vertices in the graph, if the graph is traversed using **Depth-First Traversal** algorithm. Assume vertex 0 as the starting vertex.

(C) Find the **minimum spanning tree** for the following graph. Assume vertex 0 as the source vertex. Show all steps clearly.



Question 6 [8 Marks]

Given the following input keys:
42, 52, 84, 41, 15, 30, 61
And, hash function h(X) = X mod 11, HTSize = 11 and bucket size = 1.
Obtain the resulting hash table when open addressing with **quadratic probing** is used to resolve collisions.

University of Bahrain
College of Information Technology
Department of Computer Science
Second Semester, 2010-2011
ITCS215 (Data Structures)

Final Exam

Question 1 [6 + 6 Marks]

(A) **[6 Marks]** Write member function insertLast for class **linkedListType** (as discussed in the lectures) to insert a new node at the end of the linked list.

Function prototype:

```
void insertLast (const Type& newItem);
```

(B) **[6 Marks]** Write a member function splitList( ) to be included in class **linkedListType** that takes an object of type **linkedListType** as parameter. The function splits the given list by deleting the alternative elements from the given list and inserting them into list2.

- The function should keep the same order of the elements in both lists.
- The function should print an error message if the given list is empty.

Example:

Before calling splitList( ): given list ("this object"): 1 2 3 4 5 6 7

After calling splitList( ): given list ("this object"): 1 3 5 7 and list2: 2 4 6

Function prototype:

```
void splitList (linkedListType<Type>& list2);
```

Note: You can call member function insertLast of part(A) and also deleteNode member function of class **linkedListType**. Prototype of deleteNode member function is as follows:

```
void deleteNode (const Type& deleteItem);

// This function deletes the node with info as deleteItem
```

Question 2 [14 Marks]

(A) **[8 Marks]** Write a function (not member function) called `separatePosNeg` that accepts two stacks `S1` and `S2`, consisting of numbers, as parameters. The function will remove all positive numbers from `S1` and insert them in `S2` and will also remove all negative numbers from `S2` and insert them in `S1`. Zero's in both stacks should not move. In other words, `S1` will hold all negative numbers and `S2` will hold all positive numbers. Use common stack operations such as ***push***, ***pop***, ***top***, ***isEmptyStack, isFullStack***, ***copy constructor*** and ***operator=*** function only.

Function prototype:

```
void    separatePosNeg(stackType<Type>& S1, stackType<Type>& S2);
```

(B) [6 Marks] Evaluate the following postfix expression using stacks, Show all steps clearly.

5 6 2 * 3 / 2 * 6 - *

Question 3 [14 Marks]

(A)[8 Marks] Write a non-member function called **swapPartQueue** that receives two parameters: an object called Q of type **QueueType** and an integer called **rank**. The function takes all the items found after the rank-th position and place them at the front of object Q. if Q is empty or the **rank** is out of range then the function returns false, else it returns true.

Function prototype:

bool swapPartQueue (QueueType<Type> &Q, int rank);

Assume that the class **QueueType** is available for use. Use only the member functions of class **QueueType** such as copy constructor, overloaded assignment operator, isEmptyQueue, isFullQueue, front, back, addQueue and deleteQueue.

For example,

Q

	queueFront							queueRear				
	A	B	C	D	E	F	G	H				

After calling swapPartQueue(Q, 4);

	queueFront							queueRear	
. . .	E	F	G	H	A	B	C	D	…

(B))[6 marks] What is the output of the following program:

```
#include <iostream>
#include <stackType.h>
#include<queueType.h>

int main( )
{
    stackType<int>   S;
    queueType<int>   Q;

    for (int i = 1; i<20 ; i+=3 )
       Q.addQueue( i );

    while ( !Q.isEmptyQueue() )
    {
       if ( Q.Front() % 2 == 0 )
          S.push ( Q.front() );
       else
          S.push( Q.front() * 2 );
       Q.deleteQueue();
    }

    int sum = 0;
    while ( !S.isEmptyStack() )
```

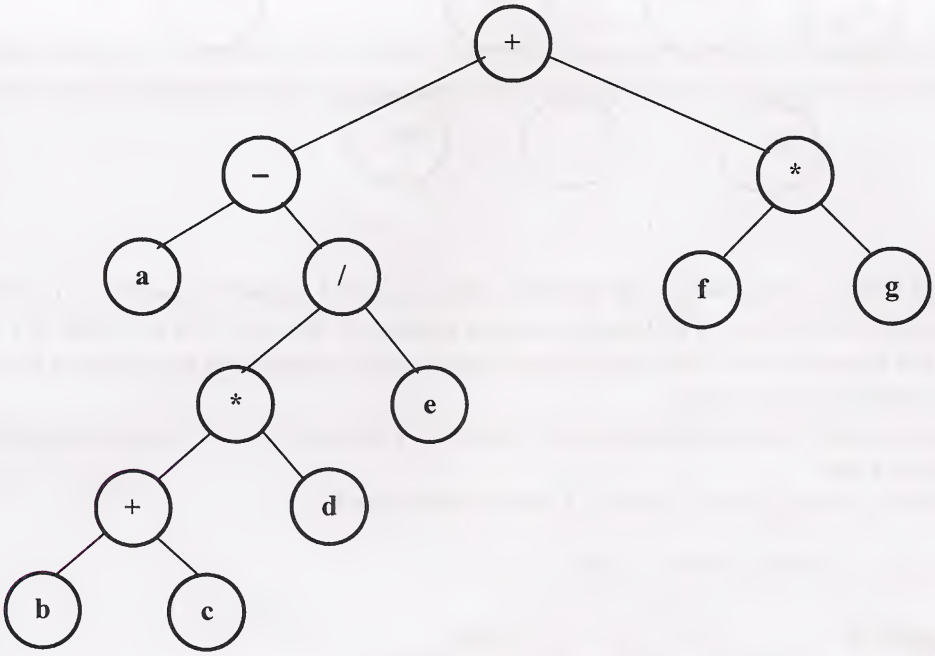
```
        {
            if ( S.top() <= 20 )
                cout<< S.top() << “ ”;
            else
                sum += S.top();
            S.pop();
        }

        cout<<”\n Sum=  “<< sum << endl;
        return 0;
    }
```

Write the output in the box below:

Question 4 [10 + 5 + 7 Marks].

(A) For the binary tree given below answer the following questions:

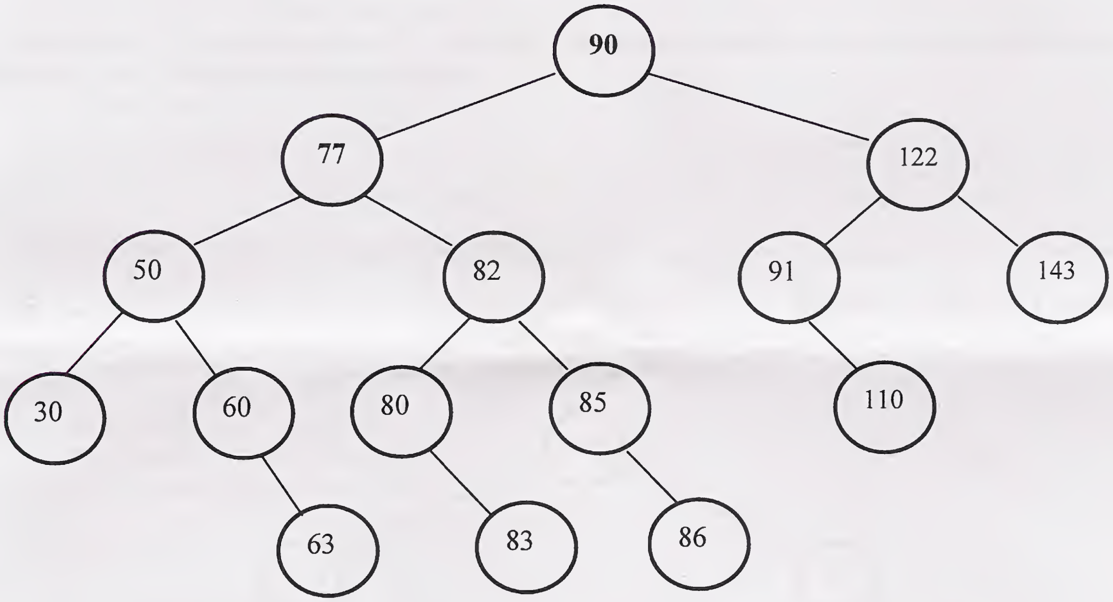


(i) **[2 mark]** What is the height of this binary Tree?

(ii) **[3 marks]** List the nodes of the right sub-tree.

(iii) **[5 marks]** Find the sequence of nodes, if the binary tree is traversed in post-order traversal.

(B) **[5 marks]** Redraw the following binary search tree after deleting the nodes **60**, **90** consecutively.



(B) [7 marks] Write a recursive private member function called **`sumWithLeft`** to be included in class `binaryTreeType`. The function returns the sum of the `info` of all nodes in a binary tree whose left sub-tree is not null and right sub-tree is null. Assume that the nodes of the binary tree contain numbers as the `info`.

This function is called from a public member function `treesumWithLeft`, given as follows:

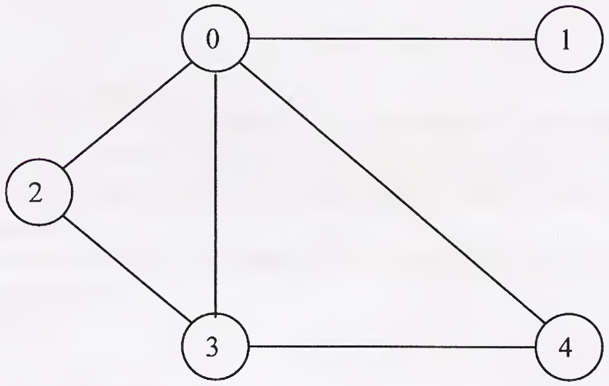
```
template<class Type>
Type  binaryTreeType<Type>::treesumWithLeft( )
{
      return  sumWithLeft(root );
}
```

Function Prototype:

```
   Type  sumWithLeft(nodeType<Type> *p);
```

Question 5 [10 + 8 Marks]

(A) [5 + 5 Marks] For the following graph, find the **adjacency matrix** and **adjacency list** representation of the graph:



(B)[8 Marks] For the graph shown below, find the sequence of vertices in the graph, if the graph is traversed using **Breadth-First Traversal** algorithm. Assume vertex 0 as the starting vertex.